

# Потоки

Пакет **java.io** содержит **классы ввода/вывода**, который может потребоваться для совершения **ввода** и **вывода** в Java.

Поток в пакете **java.io** осуществляет поддержку различных данных, таких как примитивы, объекты, локализованные символы и т.д.

**Потоки в Java** определяются в качестве **последовательности данных**. Существует два типа потоков:

- **InPutStream** – поток ввода используется для считывания данных с источника.
- **OutPutStream** – поток вывода используется для записи данных по месту назначения.

Java предоставляет сильную, но гибкую поддержку в отношении **ввода/вывода**, связанных с файлами и сетями.

## Байтовый

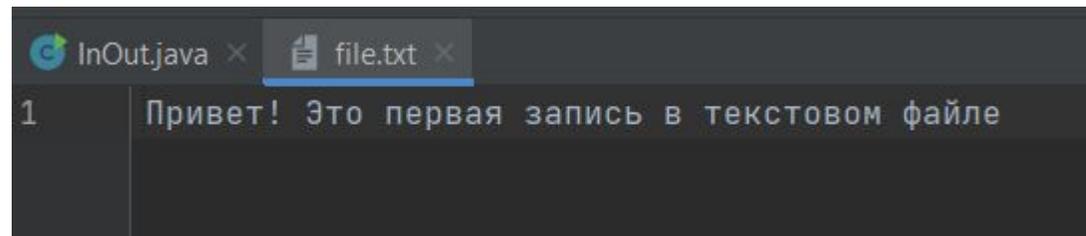
**поток** Потоки байтов в Java используются для осуществления ввода и вывода 8-битных байтов. Не смотря на множество классов, связанных с потоками байтов, наиболее распространено использование следующих классов: **FileInputStream** и **FileOutputStream**.

Пример копирования данных в

**Примечание:** чтобы скопировать файл, необходимо в папке проекта создать файл file.txt с любым или пустым содержимым.

```
FileOutputStream fileOutputStream = new FileOutputStream(  
    name: "C:\\Users\\nastj\\IdeaProjects\\Date\\src\\file.txt");  
//Указываем путь к документу в который будет записана информация  
String greetings = "Привет! Это первая запись в текстовом файле";  
fileOutputStream.write(greetings.getBytes());  
fileOutputStream.close();  
}
```

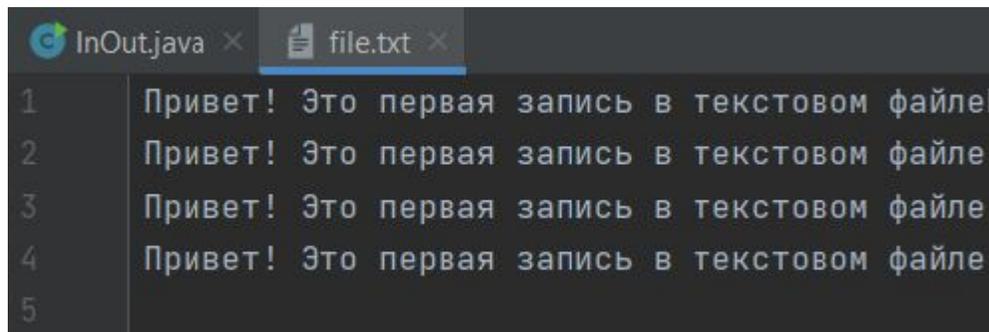
**РЕЗУЛЬТ  
АТ**



```
InOut.java × file.txt ×  
1 Привет! Это первая запись в текстовом файле
```

```
static void InOut1() throws IOException {  
  
    FileOutputStream fileOutputStream = new FileOutputStream(  
        name: "C:\\Users\\nastj\\IdeaProjects\\Date\\src\\file.txt", append: true);  
    //Указываем путь к документу в который будет записана информация  
    String greetings = "Привет! Это первая запись в текстовом файле\n";  
    fileOutputStream.write(greetings.getBytes());  
    fileOutputStream.close();  
}
```

## РЕЗУЛЬТ АТ



```
InOut.java x file.txt x  
1 Привет! Это первая запись в текстовом файле  
2 Привет! Это первая запись в текстовом файле  
3 Привет! Это первая запись в текстовом файле  
4 Привет! Это первая запись в текстовом файле  
5
```

Не забываем об освобождении ресурсов через метод **close()**

# Класс

## OutputStream

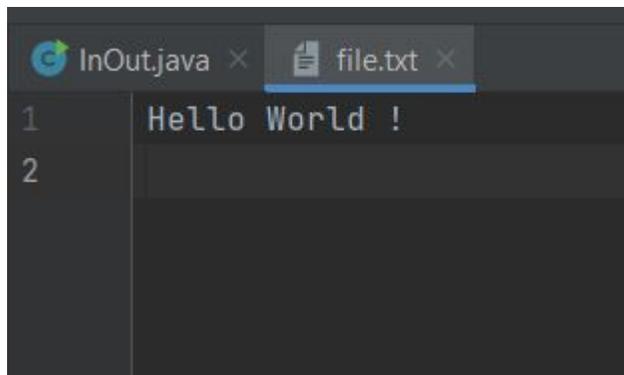
Класс OutputStream является базовым классом для всех классов, которые работают с бинарными потоками записи. Свою функциональность он реализует через следующие методы:

- `void close()`: закрывает поток
- `void flush()`: очищает буфер вывода, записывая все его содержимое
- `void write(int b)`: записывает в выходной поток один байт, который представлен целочисленным параметром `b`
- `void write(byte[] buffer)`: записывает в выходной поток массив байтов `buffer`.
- `void write(byte[] buffer, int offset, int length)`: записывает в выходной поток некоторое число байтов, равное `length`, из массива `buffer`, начиная со смещения `offset`, то есть с элемента `buffer[offset]`.

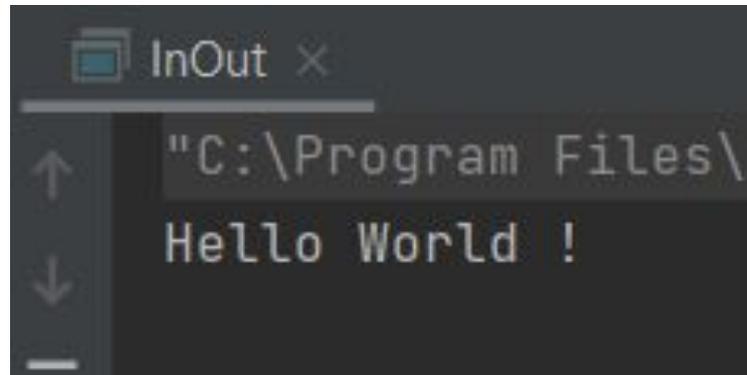
## Пример чтения данных из

```
static void InOut2() throws IOException{  
    FileInputStream fileInputStream = new FileInputStream(  
        name: "C:\\Users\\nastj\\IdeaProjects\\Date\\src\\file.txt");  
    int i;  
    while((i=fileInputStream.read())!= -1){  
        System.out.print((char)i);  
    }  
}
```

**РЕЗУЛЬТ  
АТ**



The screenshot shows an IDE window with two tabs: 'InOut.java' and 'file.txt'. The 'file.txt' tab is active, displaying the text 'Hello World !' on line 1. Line 2 is empty.



The screenshot shows a terminal window titled 'InOut'. The output of the program is displayed as 'C:\\Program Files\\' on the first line and 'Hello World !' on the second line.

\*InputStream символы не читают, так что писать на русском нельзя, для этого используются ридеры.

# Класс

## InputStream

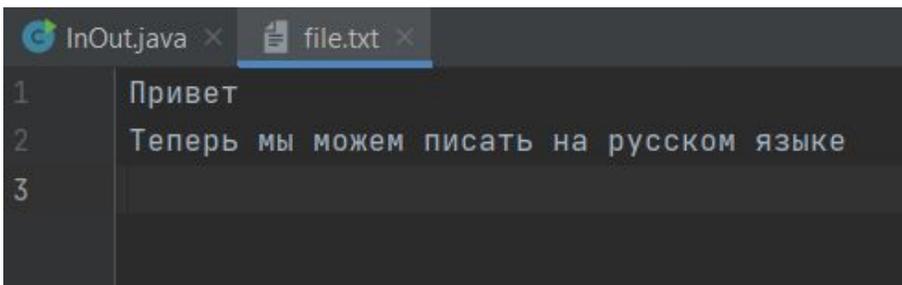
Класс `InputStream` является базовым для всех классов, управляющих байтовыми потоками ввода. Рассмотрим его основные методы:

- `int available()`: возвращает количество байтов, доступных для чтения в потоке
- `void close()`: закрывает поток
- `int read()`: возвращает целочисленное представление следующего байта в потоке. Когда в потоке не останется доступных для чтения байтов, данный метод возвратит число -1
- `int read(byte[] buffer)`: считывает байты из потока в массив `buffer`. После чтения возвращает число считанных байтов. Если ни одного байта не было считано, то возвращается число -1
- `int read(byte[] buffer, int offset, int length)`: считывает некоторое количество байтов, равное `length`, из потока в массив `buffer`. При этом считанные байты помещаются в массиве, начиная со смещения `offset`, то есть с элемента `buffer[offset]`. Метод возвращает число успешно прочитанных байтов.
- `long skip(long number)`: пропускает в потоке при чтении некоторое количество байт, которое равно `number`

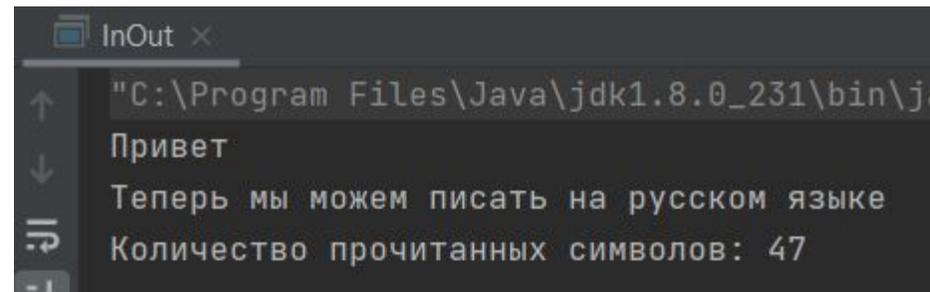
```
static void InOut3(){
    char[] array = new char[50];
    int size = 0;
    File file = new File(
        pathname: "C:\\Users\\nastj\\IdeaProjects\\Date\\src\\file.txt");
    try (FileWriter fw = new FileWriter(file);
        FileReader fr = new FileReader(file)) {

        fw.write(str: "Привет\nТеперь мы можем писать на русском языке\n");
        fw.flush();
        size = fr.read(array);

        for (int i = 0; i < size; i++) {
            System.out.print(array[i]);
        }
    } catch (IOException e) {
        System.out.print(e.getMessage());
    }
    System.out.println("Количество прочитанных символов: "
        + size + " ");
}
```



```
InOut.java x file.txt x
1 Привет
2 Теперь мы можем писать на русском языке
3
```



```
InOut x
C:\Program Files\Java\jdk1.8.0_231\bin\java.exe
Привет
Теперь мы можем писать на русском языке
Количество прочитанных символов: 47
```

**РЕЗУЛЬТ  
АТ**

# Абстрактные классы Reader и Writer

Абстрактный класс Reader предоставляет функционал для чтения текстовой информации. Рассмотрим его основные методы:

- `abstract void close()`: закрывает поток ввода
- `int read()`: возвращает целочисленное представление следующего символа в потоке. Если таких символов нет, и достигнут конец файла, то возвращается число -1
- `int read(char[] buffer)`: считывает в массив `buffer` из потока символы, количество которых равно длине массива `buffer`. Возвращает количество успешно считанных символов. При достижении конца файла возвращает -1
- `int read(CharBuffer buffer)`: считывает в объект `CharBuffer` из потока символы. Возвращает количество успешно считанных символов. При достижении конца файла возвращает -1
- `abstract int read(char[] buffer, int offset, int count)`: считывает в массив `buffer`, начиная со смещения `offset`, из потока символы, количество которых равно `count`
- `long skip(long count)`: пропускает количество символов, равное `count`. Возвращает число успешно пропущенных символов

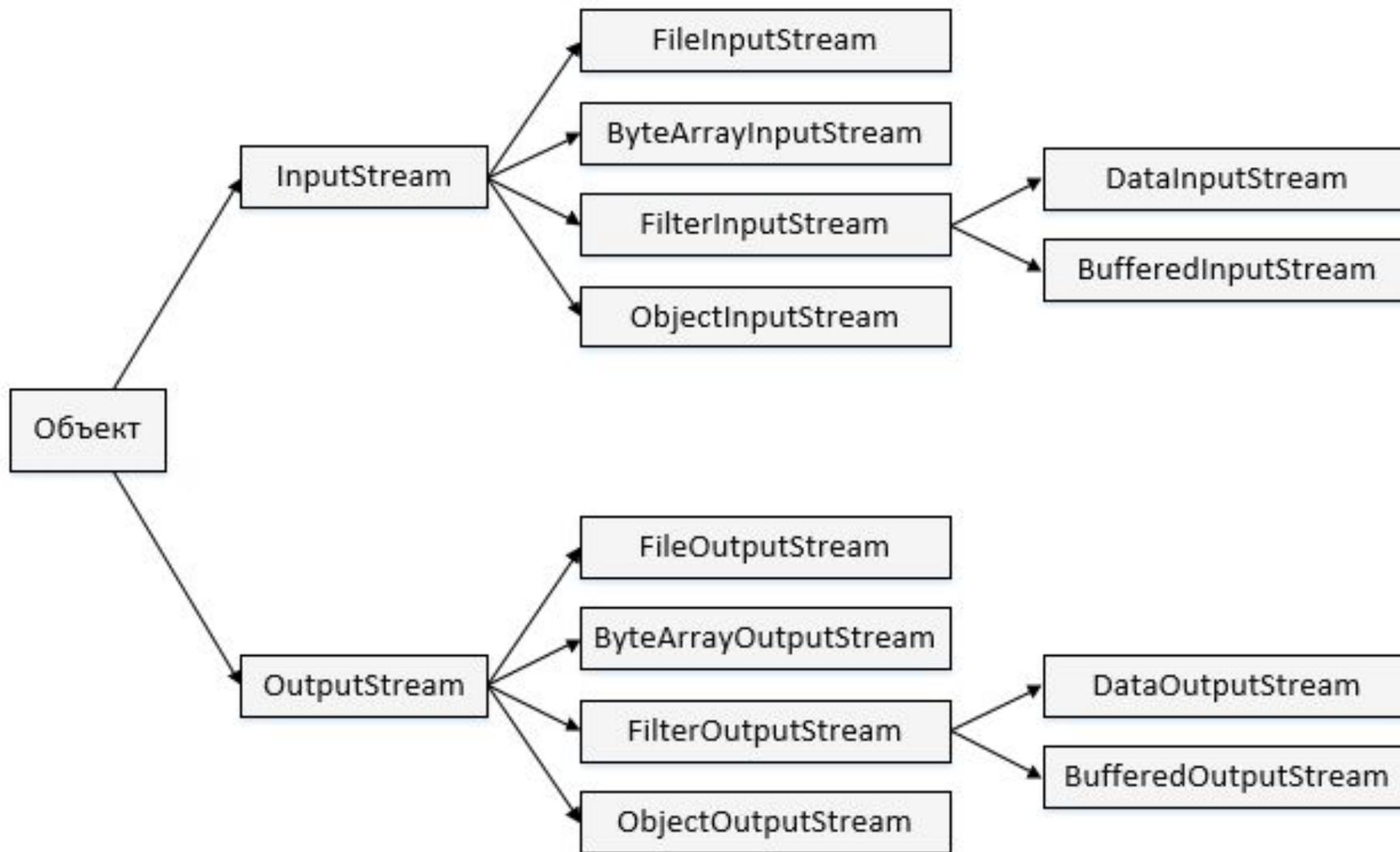
Класс `Writer` определяет функционал для всех символьных потоков вывода. Его основные методы:

- `Writer append(char c)`: добавляет в конец выходного потока символ `c`. Возвращает объект `Writer`
- `Writer append(CharSequence chars)`: добавляет в конец выходного потока набор символов `chars`. Возвращает объект `Writer`
- `abstract void close()`: закрывает поток
- `abstract void flush()`: очищает буферы потока
- `void write(int c)`: записывает в поток один символ, который имеет целочисленное представление
- `void write(char[] buffer)`: записывает в поток массив символов
- `abstract void write(char[] buffer, int off, int len)` : записывает в поток только несколько символов из массива `buffer`. Причем количество символов равно `len`, а отбор символов из массива начинается с индекса `off`
- `void write(String str)`: записывает в поток строку
- `void write(String str, int off, int len)`: записывает в поток из строки некоторое количество символов, которое равно `len`, причем отбор символов из строки начинается с индекса `off`

Функционал, описанный классами `Reader` и `Writer`, наследуется непосредственно классами символьных потоков, в частности классами `FileReader` и `FileWriter` соответственно, предназначенными для работы с текстовыми файлами.

# Чтение и запись файла

Иерархия классов для управления потоками Ввода и Вывода



# Класс

## BufferedInputStream

Буферизированные потоки нужны прежде всего для оптимизации ввода-вывода.

Обращение к источнику данных, например, чтение из файла, — дорогостоящая в плане производительности операция. И каждый раз обращаться к файлу для чтения по одному байту расточительно.

Поэтому **BufferedInputStream** считывает данные не по одному байту, а блоками и временно хранит их в специальном буфере. Это позволяет оптимизировать работу программы за счет того, что уменьшается количество обращений к файлу.

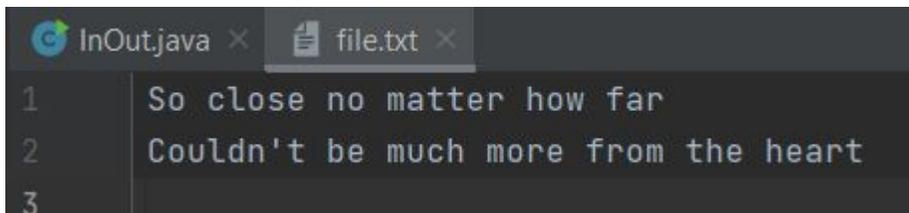
```
static void InOut4() throws IOException {
    FileInputStream fileInputStream = new FileInputStream(
        name: "C:\\Users\\nastj\\IdeaProjects\\Date\\src\\file.txt");

    BufferedInputStream bufferedInputStream = new BufferedInputStream(fileInputStream, size: 200);
    int i;
    while((i = bufferedInputStream.read()) != -1){

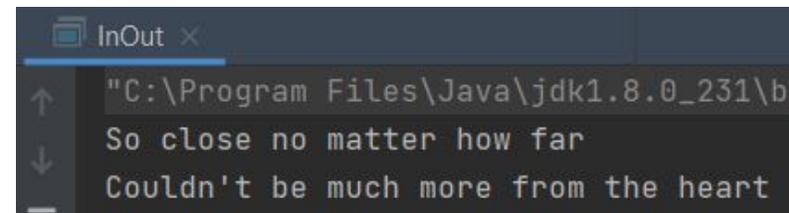
        System.out.print((char)i);
    }
}
```

размер  
буфера  
в байтах.

РЕЗУЛЬТ  
АТ



```
InOut.java x file.txt x
1 So close no matter how far
2 Couldn't be much more from the heart
3
```



```
InOut x
↑ "C:\Program Files\Java\jdk1.8.0_231\bi
↓ So close no matter how far
_ Couldn't be much more from the heart
```

```
static void InOut5() throws IOException {

    Date date = new Date();
    FileInputStream fileInputStream = new FileInputStream(
        name: "C:\\Users\\nastj\\IdeaProjects\\Date\\src\\file.txt");
    BufferedInputStream bufferedInputStream = new BufferedInputStream(fileInputStream);
    int i;
    while((i = bufferedInputStream.read()) != -1){ System.out.print((char)i); }
    Date date1 = new Date();
    int a = (int) (date1.getTime() - date.getTime());
    System.out.println("\nВремя считывания информации в буфере = " + a);

    Date date_2 = new Date();
    FileInputStream fileInputStream1 = new FileInputStream(
        name: "C:\\Users\\nastj\\IdeaProjects\\Date\\src\\file.txt");
    int i1;
    while((i1 = fileInputStream1.read()) != -1){ /*System.out.print((char)i1);*/ }
    Date date2 = new Date();
    int b = (int) (date2.getTime() - date_2.getTime());
    System.out.println("\nВремя считывания информации по байтам = " + b);
}
```

# РЕЗУЛЬТ

АТ

```
InOut.java x file.txt x
1 Couldn't be much more from the heart
2 Couldn't be much more from the heart
3 Couldn't be much more from the heart
4 Couldn't be much more from the heart
5 Couldn't be much more from the heart
6 Couldn't be much more from the heart
7 Couldn't be much more from the heart
8 Couldn't be much more from the heart
9 Couldn't be much more from the heart
10 Couldn't be much more from the heart
```

```
InOut x
"C:\Program Files\Java\jdk1.8.0_231\bin\java.exe"
Couldn't be much more from the heart
Время считывания информации в буфере = 4
Couldn't be much more from the heart
Couldn't be much more from the heart
Couldn't be much more from the heart
Время считывания информации по байтам = 7
```

## Задание: Опросник

### Требования:

- Вопросы берутся из текстового файла
- Ответы сохраняются в текстовом файле
- Минимум 5 вопросов и ответов
- По итогу прохождения теста, в текстовом файле отображается время за которое пользователь ответил на все вопросы (например начало и конец работы)
- В текстовом файле должна отображаться дата прохождения теста